Noname manuscript No.
(will be inserted by the editor)

Computing strong lower and upper bounds for the integrated multiple-depot vehicle and crew scheduling problem with branch-and-price

Markó Horváth · Tamás Kis

Received: date / Accepted: date

Abstract In the problem of the title, vehicle and crew schedules are to be determined simultaneously in order to satisfy a given set of trips over time. The vehicles and the crew are assigned to depots, and a number of rules have to be observed in the course of constructing feasible schedules. The main contribution of the paper is a novel mathematical programming formulation which combines ideas from known models, and an exact solution procedure based on branch-and-price. The method is tested on benchmark instances from the literature and it provides suboptimal schedules using limited computational resources.

Keywords vehicle and crew scheduling · branch-and-price · exact methods · integer programming

1 Introduction

The vehicle scheduling and the crew scheduling problems are two main planning problems that arise in the operational phase of the planning process of public transport companies, and have several real-world applications, e.g., at the public transport company of Rotterdam, the Netherlands (Huisman, 2004), in Ljubljana, Republic of Slovenia (Békési et al., 2009) and in Szeged, Hungary (Balogh and Békési, 2014). Briefly stated, the aim of these problems is to find an

Markó Horváth
Institute for Computer Science and Control, Hungarian Academy of Sciences, H1111 Budapest, Kende str. 13–17, Hungary

Tamás Kis
Institute for Computer Science and Control, Hungarian Academy of Sciences, H1111 Budapest, Kende str. 13–17, Hungary
Tel.: +36-1-279-6156
Fax: +36-1-466-7503
E-mail: tamas.kis@sztaki.mta.hu

assignment of minimum cost of a given set of trips to vehicles, and to create a minimal cost set of crew duties that cover tasks resulted from vehicle schedules. In the traditional sequential approach, the vehicle scheduling problem is solved first and then the crew scheduling problem next, but Bodin et al. (1983) criticize scheduling vehicles independently of the crew, because in the mass transit case crew costs mostly dominate vehicle operating costs. The *integrated vehicle and crew scheduling problem* aims to schedule vehicles and the crew simultaneously, rather than sequentially.

In this paper we describe a novel mathematical programming formulation for the integrated multiple-depot vehicle and crew scheduling problem, where we combine the advantages of the existing modeling approaches. While most of the known MIP formulations model the vehicle and crew schedules separately, and join the two parts by linking constraints, we model crew schedules along with some extra variables and constraints that ensure that from any integer feasible solution a valid vehicle schedule can be deduced as well. Further on, any optimal solution of our MIP formulation represents an optimal solution for the integrated vehicle and crew scheduling problem. Our modeling approach is quite general, the set of columns represents the valid crew schedules, and a subset of it is generated in the course of the solution procedure guided by the rules to be observed by valid driver schedules. We also present our exact branch-and-price procedure for this formulation, where we develop an efficient variable pricing method, some branching rules, and we apply several acceleration strategies. We test our approach on well-known problem instances.

To our best knowledge, the only paper proposing an exact method for the integrated multiple-depot vehicle and crew scheduling problem is that of Mesquita et al. (2009), where a variant of the problem is studied in which some of the common assumptions we and other authors make on feasible crew schedules are neglected. Their MIP formulation, unlike ours, models vehicle and crew schedules separately and contains additional linking constraints to join the two parts.

This paper is organized as follows. In Section 2 we give a formal problem statement along with the assumptions on the input and admissible solutions. We review the related literature in Section 3, where we also highlight the novelty of our approach. In Section 4 we present our problem formulation, and we describe our solution method in Section 5. We summarize our computational results in Section 6, and conclude the paper in Section 7.

2 Problem definition

A *trip* is a project for vehicles to carry passengers between two given stations, and we assume that each trip is timetabled, that is, it has fixed departure and arrival time. A fleet of vehicles may consist of different vehicle types, and some trips may not be operated by all vehicle types. Thus, although a depot basically is a storage facility, where vehicles can be parked when not in use, we treat a *depot* as a facility with homogeneous fleet of vehicles (that is, if such a facility

consists of several vehicle types, we partition its inhomogeneous fleet into homogeneous ones). The *vehicle scheduling problem* (VSP) can be stated as follows: we are given a set of trips, a fleet of vehicles divided into depots and the goal is to find an assignment of trips to vehicles such that each trip is assigned exactly once; each vehicle performs a feasible sequence of trips; each sequence starts and ends at the same depot; and asset and operational costs are minimized. Based on the number of depots, we have the *single-depot vehicle scheduling problem* (SDVSP), or the *multiple-depot vehicle scheduling problem* (MDVSP).

A *vehicle itinerary* describes the route of a vehicle, i.e., the movements made by the vehicle, e.g., performing a trip, waiting at a station or in a depot, pulling out from/pulling in a depot, performing a deadhead (that is, traveling between stations without passengers). Each vehicle itinerary starts with a *pull-out* and ends with a *pull-in*, but vehicles can return to the depot at any time. A *vehicle block* is the part of the vehicle itinerary between a pair of consecutive pull-out and pull-in (both included). In Figure 1 we depict a vehicle itinerary consisting of two vehicle blocks. Some vehicle movements require driver attendance (e.g., performing a trip/deadhead or pulling out from/pulling in a depot), while typically no driver is required to be present if the vehicle is waiting in a depot. Drivers can board/leave the vehicle only at *relief points*, these are the depots and certain designated stations. Moreover, each trip has at most two relief points: one at the beginning and one at the end of the trip, i.e., drivers cannot board/leave the vehicle while it is performing a trip. According to these restrictions, each vehicle itinerary defines tasks that have to be assigned to drivers. More precisely, a *task* is a sequence of driver requiring vehicle movements between two consecutive relief points, i.e., tasks are the most elementary portion of work that can be assigned to a driver. For example, in Figure 1 we present a situation, where a driver is required to be present if a vehicle is outside of the depot, and the only relief point other than the depot is station C. Thereby, vehicle block I and vehicle block II consist of 3 and 1 tasks, respectively. A *piece of work* is a sequence of tasks without any break (i.e., each task in a piece of work begins at the time point when the previous one ends), and a *(driver) duty* is either a single piece of work or a sequence of pieces of work separated by breaks. The first three tasks in Figure 1 could define six pieces of work (these are (task I), (task II), (task III), (task I, task II), (task II, task III) and (task I, task II, task III)), while task IV can be contained by only one piece of work. In this figure we depict only three pieces of work. Again, these three pieces of work could define four driver duties (these are (piece of work I), (piece II), (piece III), and (piece II, piece III)), however we depict only one.

The *crew scheduling problem* (CSP) can be stated as follows: find a set of duties for a given set of tasks such that each task is covered by a duty that can be performed by a single driver; each duty satisfies a wide variety of federal laws, safety regulations, and (collective) in-house agreements; and labor costs are minimized.

Finally, the *integrated vehicle and crew scheduling problem* (VCSP) can be stated as follows: for a set of trips find a minimum cost set of vehicle itineraries and driver duties such that both

the vehicle and the crew schedules are feasible and compatible with each other (that is, the driver schedule is feasible according to tasks determined by the vehicle schedule). Again, based on the number of depots we have the *single-depot vehicle and crew scheduling problem* (SDVCSP), or the *multiple-depot vehicle and crew scheduling problem* (MDVCSP).

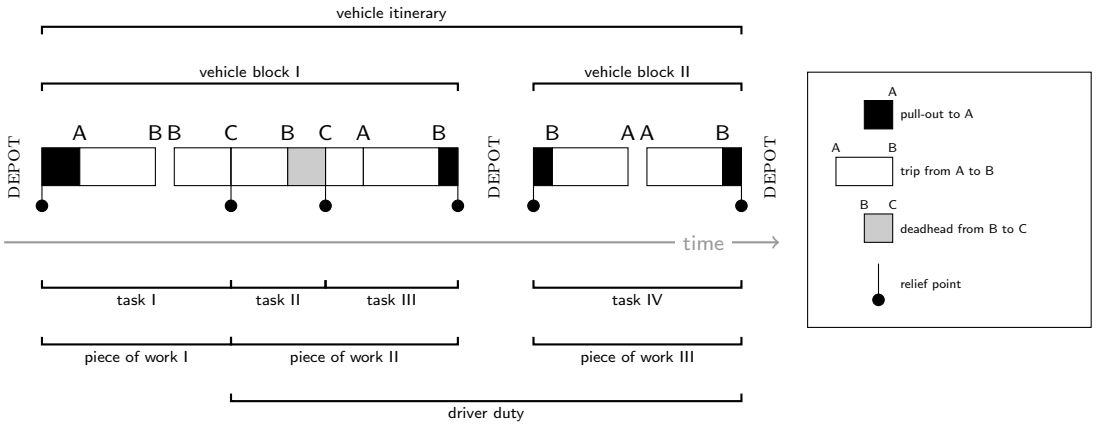

Fig. 1 Route of a vehicle and some driver activities (based on Figure 1.4 in (Steinzen, 2007))

2.1 Assumptions

In the followings we introduce our assumptions about the MDVCSP.

Rule 1 *Each vehicle is assigned to a depot where its daily schedule starts end ends. Each depot is unlimited in capacity, that is, it can store an unlimited number of vehicles.*

Rule 2 *A vehicle returns to its depot if the idle time between two consecutive trips is long enough to perform a round trip to the depot.*

Rule 3 *Each driver is assigned to a depot and may only conduct tasks on vehicles from this particular depot. However, a duty does not necessarily start and end in this depot.*

Rule 4 *A piece of work is only restricted by its duration. It may have a minimum and maximum duration.*

Rule 5 (continuous attendance) *A driver is required to be present if a vehicle is outside of a depot, while no driver is needed when the vehicle is parked in the depot.*

Rule 6 (restricted changeover) *Drivers may only change their vehicle during a break, i.e., between two pieces of work.*

Rules 1–6 are customary assumptions in the literature (Huisman, 2004; Huisman et al., 2005; Steinzen, 2007; Steinzen et al., 2010).

Rule 2 was originally proposed for vehicle scheduling problems to reduce the number of constraints by introducing the concept of short arcs, and long arcs, see e.g., Freling et al. (1995b). Basically, in their network model arcs representing vehicle movements with appropriate long idle time were replaced with so-called long arcs representing round trips to the depot, and for such arcs they did not require the continuous attendance. This idea was applied for integrated problems as well (e.g., Freling et al. (2003); Huisman et al. (2005); Steinzen et al. (2010)), however, it is worth mentioning that omitting such long waiting and deadheads may change the set of potential tasks (see Rule 5), hence the set of feasible duties can be changed. Rule 2 can create another problem when time-space network approaches are used for the VCSP. Steinzen (2007) and Steinzen et al. (2010) suggested to eliminate appropriate (long) arcs from network to ensure Rule 2, but it is not sufficient by itself as we will show in Section 4.1. That is why we will handle Rule 2 as a lazy rule, i.e., we will eliminate long arcs from the network model of the problem, but we will not make further efforts to satisfy Rule 2.

To ensure Rule 6, we need to redefine the concept of a piece of work, that is, in the rest of this paper a piece of work is a sequence of tasks without any break that is performed by the same vehicle. Remark that pieces of work in Figure 1 correspond to the new concept.

Rule 7 *A duty consists of one or two pieces of work. Each duty starts with a sign-on and ends with a sign-off by the driver. Feasibility of a duty can depend only on earliest/latest (sign-on) start/(sign-off) end time; minimum/maximum piece length; minimum/maximum break length; minimum/maximum working time; minimum/maximum spread time.*

In our terminology *working time* is the time that driver spends on the vehicle (i.e., the total duration of the pieces of work consisted by the duty), and *spread time* is the total duration of the sign-on, the sign-off, the pieces of work and the breaks.

Rule 8 *Vehicle cost is a combination of a fixed asset cost for using the vehicle and a variable operation cost. Asset cost depends only on depot. Operation cost is a linear function of travel and idle time outside the depot.*

Rule 9 *Duty cost is a combination of a fixed driver cost for using a driver and a variable working cost. Driver cost depends only on depot. Working cost is a linear function of working time.*

In fact, fixed costs in Rules 8–9 are not restrictions as we assumed that each depot consists of a homogeneous fleet of vehicles and crew is a group of anonymous drivers.

3 Literature review

3.1 Sequential vehicle and crew scheduling

The MDVSP is shown to be NP-hard by Bertossi et al. (1987), which is in strong contrast with the polynomial solvability of the SDVSP, see e.g., Freling et al. (2001). An overview of different vehicle scheduling models can be found in (Bunte and Kliewer, 2009). For heuristic solution approaches for the MDVSP we refer to Pepin et al. (2006).

Both the VSP and the CSP can be interpreted as an assignment problem, the CSP is more complicated than the VSP because of the wide variety of working rules (e.g., minimum/maximum working time for drivers, minimum/maximums spread time for duties, etc.). Fischetti et al. (1987, 1989) show that the CSP is NP-hard if either spread time or working time constraints are present.

3.2 Partial integration

Until the late nineties the complete integration of vehicle scheduling and crew scheduling was computationally intractable, thus most of the early approaches are based on a heuristic integration.

Ball et al. (1983) propose the first partially integrated approach for the single-depot case. They schedule crew first including vehicle scheduling considerations and construct a feasible vehicle schedule afterward. Similar heuristics for the single-depot case are proposed by Tosini and Vercellis (1988), Falkner and Ryan (1992), and Patrikalakis and Xerocostas (1992).

Other approaches schedule vehicles first but include crew scheduling considerations and subsequently generate feasible crew schedules, see e.g., Scott (1985) and Darby-Dowman et al. (1988).

Gintner et al. (2008) apply another partial integration approach for the multiple-depot case. They perform vehicle scheduling first and crew scheduling afterward, but they use a time-space network approach for vehicle scheduling that allow to change the corresponding optimal vehicle schedule without loss of optimality in the crew scheduling phase.

3.3 Complete integration

In Table 1 we collect the core of modeling and solution approaches of completely integrated models, details are explained below.

3.3.1 Single-depot case

Freling et al. (1995a) propose the first fully integrated approach for the single-depot case. Their integer programming model uses a so-called connection-based network and consists of three components: a quasi-assignment formulation for vehicle scheduling, a set partitioning formulation

Table 1 Modeling and solution approaches for the complete integration

Reference	Modeling approach			Solution approach	
	Network[a]	Vehicle scheduling part	Crew scheduling part	Type[b]	Core
Single-depot case					
Freling et al. (1995a)	CB	quasi-assignment	set partitioning	H	LR-CG[c]
Friberg and Haase (1999)	CB	set partitioning	set partitioning	E	BCP[d]
Haase et al. (2001)	DB	side constraints	multicommodity flow	E	BCP
Freling et al. (2003)	CB	quasi-assignment	set partitioning	H	LR-CG
Laurent and Hao (2008)		(constraint programming approach)		H	GRASP[e]
Multiple-depot case					
Gaffi and Nonato (1999)	CB	quasi-assignment	set partitioning	H	LR-CG
Huisman et al. (2005)	CB	multicommodity flow	set partitioning	H	LR-CG
Borndörfer et al. (2008)	CB	multicommodity flow	set partitioning	H	LR-CG
Mesquita and Paias (2008)	CB	multicommodity flow	set partitioning/covering	H	PB[f]
Mesquita et al. (2009)	CB	multicommodity flow	set partitioning/covering	H/E	BP[g]
Steinzen et al. (2010)	TS	multicommodity flow	set partitioning	H	LR-CG

[a] CB: connection-based; DB: driver-based; TS: time-space
[b] H: heuristic approach; E: exact method
[c] Lagrangian relaxation based column generation
[d] branch-and-cut-and-price
[e] greedy randomized adaptive search procedure
[f] (LP-relaxation based) price-and-branch
[g] branch-and-price

for crew scheduling, and additional linking constraints that ensure the compatibility of vehicle and crew schedules. Their solution approach uses column generation in combination with Lagrangian relaxation. That is, linking constraints are relaxed in a Lagrangian way and the crew scheduling part is relaxed to a set covering formulation that yields two independent Lagrangian subproblems: a single-depot vehicle scheduling problem and a selection problem. They solve the Lagrangian dual problem with a subgradient algorithm, and suggest a two-phase pricing method to generate new columns (i.e., duties) for the crew scheduling part. They apply several heuristics to obtain feasible integer solutions for the original problem. This modeling and solution approach provides the basis for many other publications, e.g., Freling et al. (2003); Huisman (2004); Huisman et al. (2005); Steinzen (2007); Steinzen et al. (2010).

Friberg and Haase (1999) propose the first exact algorithm for the single-depot case. Their mathematical programming formulation is a combination of set partitioning formulations for the vehicle scheduling problem and for the crew scheduling problem, respectively. They develop a branch-and-cut-and-price algorithm, i.e., the LP-relaxation in each node of the search-tree is solved by column generation, moreover, polyhedral cuts are added to strengthen the relaxation. Columns for the vehicle scheduling subproblem are generated by solving shortest path problems on acyclic graphs, however, the pricing problem for the crew scheduling subproblem is modeled as a resource constrained shortest path problem which is solved by a dynamic programming algorithm.

Haase et al. (2001) propose another exact solution approach for the single-depot case. In their view each driver duty must start and end in the depot. Their crew-based mathematical model is a multicommodity flow formulation that relies on a so-called driver network structure. Side constraints are used to guarantee that an optimal compatible vehicle schedule could be derived. That formulation uses a set of path flow variables for drivers and only one additional variable to count vehicles. They propose a branch-and-price algorithm, where cutting planes are added to the master problem to reinforce linear relaxations throughout the search-tree. Each pricing problem is transformed into a shortest path problem with resource constraints and solved by a dynamic programming algorithm.

Laurent and Hao (2008) consider a situation where all vehicles are parked in the same depot, however, the vehicles may belong to different categories. Thus, their case is more general than a single-depot case, but more special than the general multiple-depot case which we consider in this paper. They also use simplified crew constraints in contrast to Rule 7, e.g., they have restrictions only for the spread and working times. Their formulation relies on a constraint satisfaction and optimization model, and they apply a heuristic greedy randomized adaptive search procedure to solve the problem.

3.3.2 Multiple-depot case

Gaffi and Nonato (1999) introduce the integrated problem for the multiple-depot case. However, their approach is developed for the extra-urban mass transit setting, where drivers are virtually tied to their vehicles. Hence, for example, they assume that a driver is assigned to the same vehicle during the whole duty, and all pieces of work start and end in the depot. Their heuristic procedure is based on column generation in combination with Lagrangian relaxation.

Huisman (2004) and Huisman et al. (2005) propose the first general approaches for the multiple-depot case. Huisman (2004) explicitly introduces Rules 1, 3, 4, 5, 6, and Rule 2 is applied in his mathematical formulation to reduce the number of constraints. That formulation complies also with Rules 7, 8 and 9. Huisman (2004) and Huisman et al. (2005) extend the modeling and solution approaches of Freling et al. (2003) and Haase et al. (2001) for the multiple-depot case. That is, they use a multicommodity flow formulation for the vehicle scheduling part which is based on connection-based networks, and additional constraints are used to link duty and flow variables. In the first phase of their solution approach they calculate a lower bound on the optimum using a column generation algorithm where the master problem is solved with Lagrangian relaxation by a subgradient algorithm. For generating duties they apply a two-step procedure similar to that of Freling et al. (1995a), that is, they generate pieces of work with shortest path algorithms, while duties are generated by a simple enumerating procedure. Feasible solutions are obtained in the second phase. Huisman (2004) and Huisman et al. (2005) propose an alternative formulation obtained from the previous one containing only variables related to crew duties.

However, additional constraints are added to count the number of vehicles and to consider fixed vehicle costs. They apply a solution approach similar to the one for the previous formulation.

In (Huisman, 2004; Huisman et al., 2005) the authors propose their randomly generated instances which are widely used in the literature (Borndörfer et al., 2008; Mesquita and Paias, 2008; Mesquita et al., 2009; Steinzen, 2007; Steinzen et al., 2010) and in this paper as well.

Borndörfer et al. (2008) use a modeling approach similar to that of Freling et al. (1995a). Their solution approach also relies on a Lagrangian relaxation based column generation procedure, but they use inexact proximal bundle method to solve Lagrangian dual problems. The bundle method is embedded in a backtracking procedure to produce an integer solution in the second phase.

Mesquita and Paias (2008) propose a modeling approach similar to that of Huisman (2004). However, there are some fundamental differences between the problem definition of Mesquita and Paias (2008) and that of Huisman (2004). For example, in (Mesquita and Paias, 2008) the authors consider each end location of a trip as a potential relief point. Moreover, they allow drivers to change vehicles whenever there is a relief point, and to use vehicles from any depot, that is, their model do not comply with Rules 3 and 6. They use a multicommodity flow formulation for the vehicle scheduling part, and set partitioning/covering formulations for the crew scheduling part. They apply a price-and-branch algorithm, that is, they solve the LP-relaxation of the problem with a column generation approach, and if the resulted optimal solution is fractional they apply a branch-and-bound procedure to obtain feasible integer solution to the problem. The pricing problems are modeled as resource constrained shortest path problems and are solved by a dynamic programming algorithm.

Mesquita et al. (2009) propose exact and non-exact branch-and-price procedures for the same problem definition and formulation as in (Mesquita and Paias, 2008).

Steinzen (2007) and Steinzen et al. (2010) use a similar modeling approach for the multiple-depot case as in (Huisman, 2004), however, their mathematical formulation is based on time-space networks. Their Lagrangian relaxation based column generation approach is also similar to that of Huisman (2004), but in their case pricing problems are modeled by resource constrained shortest path problems on time-space networks which are solved by a dynamic programming algorithm. Finally, they devise a heuristic branch-and-price procedure which alternates between vehicle and crew scheduling to obtain feasible solutions.

Our contributions In this paper we present a novel problem formulation derived from that of Steinzen et al. (2010). We developed a branch-and-price procedure including (i) an effective pricing procedure based on that of Freling et al. (1995a) using several acceleration strategies, (ii) some branching strategies (iii) and a simple primal heuristic. We also present our computational results compared with other well-known solution approaches.

As we discussed above, several problem definitions have been proposed for the (integrated) vehicle and crew scheduling problem. Because of the differences between these assumptions, fair

comparisons cannot be established between all approaches. That is, a feasible solution for a given approach may not be feasible for another one, and vice-versa. As we mentioned in Section 2.1, our assumptions comply with those of Huisman et al. (2005); Steinzen et al. (2010), however, they differ from the assumptions of Mesquita and Paias (2008); Mesquita et al. (2009).

4 Problem formulation

In this section we discuss our mathematical formulation for the MDVCSP, and we shortly present the well-known time-space network structure the formulation bases on. First of all, we remark that we use the concept of *vertex* in graph/network terminology, and we use the concept of *node* for search-trees.

4.1 Time-space network structure

In a time-space network each vertex represents a (time, space) pair (where space is either a station or the depot), and arcs represent vehicle movements. In the following we present how we build a time-space network for a given depot. For a detailed description about building time-space networks we refer to Kliewer et al. (2006).

For each trip that can be operated from the depot we add four vertices to the network representing the (departing time, departing station), (arriving time, arriving station), (pull-out time, depot) and (pull-in time, depot) pairs, respectively. Additionally, we add a *trip arc* to the network from the *departing vertex* to the *arriving vertex*, and a *pull-out arc* (*pull-in arc*) from the *pull-out vertex* (arriving vertex) to the departing vertex (*pull-in vertex*). Of course, if a vertex or a pull-in/out arc already exists we do not duplicate them (e.g., arriving vertex of trip t_3 and departing vertex of trip t_4 are the same in Figure 2).

To represent waiting at a station or in the depot we create for each space its *timeline*, that is, we collect all vertices that represent this space and sort them in increasing order according to their represented time, then we add a *waiting arc* between consecutive vertices. Let s and t be the first and last vertex of the timeline of the depot, respectively. We add an extra *circulation arc* from t to s. Note that at a station it is sufficient to start that connecting process with the first vertex that represents arriving event, since there is no reason for a vehicle to wait at a station until a trip ends there. Moreover, according to Rule 2 we do not connect consecutive vertices together if the duration of that waiting arc would not be shorter than the duration of a round trip. As you can see in Figure 2 we do not connect the arriving vertex of trip t_1 with the departing vertex of trip t_2 at station C, since there is enough time for a vehicle to perform a round trip. It is worth mentioning that both of the two waiting arcs are necessary at station B — as they ensure the connections between trips t_2 and t_4, and trips t_3 and t_5, respectively — however, a vehicle operating trips t_2 and t_5 can use these arcs to wait in station B instead

of performing a round trip as required by Rule 2. That is why we mentioned that omitting long arcs is not sufficient to satisfy Rule 2, and that is why we do not strive to satisfy Rule 2 in the rest of our solution approach.

To represent deadhead movements between stations we add *deadhead arcs* connecting the arriving vertex of a trip with the departing vertex of an another trip. One of the most important properties of time-space networks is that we should not represent all of the deadhead movements explicitly. For example, in Figure 2 trips t_3 and t_6 are compatible (i.e., can be performed by the same vehicle), thus we connect their corresponding arriving/departing vertices with a deadhead arc. However, trips t_2 and t_6 are also compatible, but is not necessary to add any deadhead arc between them, since these can be operated by the same vehicle by using the first waiting arc and the deadhead arc. Of course, we omit a deadhead arc if it is longer than the corresponding round trip.

Note that each path from s to t corresponds to a vehicle itinerary (and vice versa), and a piece of work can be represented as a path between two relief points using nondepot-arcs only.

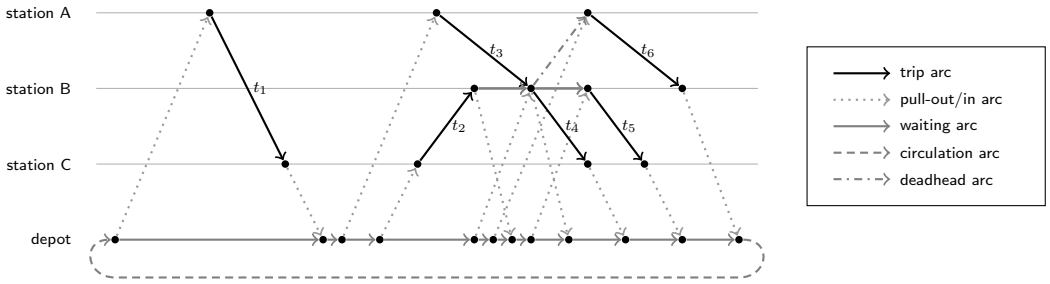

Fig. 2 Time-space network

4.2 Mathematical formulation

In this section we describe our formulation used in the rest of the paper. But before, we describe that of Steinzen et al. (2010), from which our formulation will be derived.

4.2.1 The problem formulation of Steinzen et al. (2010)

Let $\mathcal{D} = \{d_1, d_2, \ldots, d_{|\mathcal{D}|}\}$ be the set of depots, and $\mathcal{T}$ be the set of trips. Let $D^d = (V^d, A^d)$ be the time-space network for depot $d \in \mathcal{D}$, and let $\tilde{A}^d \subset A^d$ be the set of nondepot-arcs (i.e., all arcs but the arcs of the timeline of the depot and the circulation arc). It is worth mentioning that $\tilde{A}^d$ is the set of arcs that require both of vehicle and driver activities. Remember that a path between two vertices that correspond to relief points and using nondepot-arcs only represents a piece of work. Let K^d be the set of feasible duties that can be operated from depot $d \in \mathcal{D}$

and $K^d(i,j) \subseteq K^d$ the set of duties covering arc $(i,j) \in \tilde{A}^d$. For depot $d \in \mathcal{D}$ we denote by $A^d(t) \subseteq A^d$ the set of arcs corresponding to trip $t \in \mathcal{T}$. Note that $A^d(t)$ is empty if trip t cannot be operated from depot d, otherwise it contains a single arc.

Steinzen et al. (2010) use two types of variables. First, they associate a flow variable y_{ij}^d with each arc $(i,j) \in A^d$ indicating whether that arc is used and assigned to depot $d \in \mathcal{D}$. The binary duty variables x_k^d ($k \in K^d$) indicate whether duty k is selected for depot $d \in \mathcal{D}$.

On the one hand, Steinzen et al. (2010) assign a vehicle cost c_{ij}^d to each arc $(i,j) \in A^d$. That is, c_{ij}^d is the asset cost for using a vehicle if (i,j) is the circulation arc of D^d; c_{ij}^d is the operation cost of the represented vehicle movement if $(i,j) \in \tilde{A}^d$; otherwise c_{ij}^d is equal to zero. On the other hand, one could associate a working cost g_{ij}^d with each arc $(i,j) \in \tilde{A}^d$. With this, the duty cost f_k^d of duty $k \in K^d$ is the sum of the fixed driver cost, and the working cost of its pieces of work. The formulation of Steinzen et al. (2010) is the following:

$$\min \sum_{d\in\mathcal{D}} \sum_{(i,j)\in A^d} c_{ij}^d y_{ij}^d + \sum_{d\in\mathcal{D}} \sum_{k\in K^d} f_k^d x_k^d \tag{1}$$

$$\sum_{d\in\mathcal{D}} \sum_{(i,j)\in A^d(t)} y_{ij}^d = 1, \ \forall\, t \in \mathcal{T} \tag{2}$$

$$\sum_{j:(j,i)\in A^d} y_{ji}^d - \sum_{j:(i,j)\in A^d} y_{ij}^d = 0 \ \ \forall\, d \in \mathcal{D}, \forall\, i \in V^d \tag{3}$$

$$\sum_{k\in K^d(i,j)} x_k^d - y_{ij}^d = 0 \ \ \forall\, d \in \mathcal{D}, \forall\, (i,j) \in \tilde{A}^d \tag{4}$$

$$0 \le y_{ij}^d \le u_{ij}^d, \ y_{ij}^d \in \mathbb{N}, \ \forall\, d \in \mathcal{D}, \forall\, (i,j) \in A^d \tag{5}$$

$$x_d^k \in \{0,1\}, \ \forall\, d \in \mathcal{D}, \forall\, k \in K^d. \tag{6}$$

The objective (1) minimizes the sum of vehicle and crew costs. Constraint set (2) ensures that the set of trips are partitioned among the depots and each trip is covered by a single vehicle. Constraints (3) are the flow conservation constraints corresponding to the multicommodity flow formulation for the vehicle scheduling problem. Constraint set (4) links the vehicle and crew schedules, that is, each nondepot-arc should be covered by the same number of vehicles and duties. Constraints (5) ensure that the maximum capacity of flow variables is satisfied. Steinzen et al. (2010) set u_{ij}^d to 1 on trip arcs $(i,j) \in \tilde{A}^d$, however, these constraints are redundant according to (2). They also set u_{ij}^d to 1 on pull-in/out arcs $(i,j) \in \tilde{A}^d$, which are technical constraints (note that they use unique pull-in/out arcs for each trip). For all other arcs they use maximum capacity u^d equal to the number of vehicles available in depot $d \in \mathcal{D}$.

4.2.2 Our problem formulation

Our mathematical programming formulation is obtained from that of Steinzen et al. (2010) described above by dropping the redundant and technical capacity constraints from (5), and eliminating most of the flow variables by substituting them using constraints (4).

However, our formulation can also be interpreted directly from the problem definition. We use the same notations as before. Further on, let $\bar{A}^d = A^d \setminus \tilde{A}^d$ be the set of depot-arcs (i.e., the arcs of the timeline of the depot and the circulation arc), and $\bar{V}^d \subset V^d$ be the set of depot-vertices of D^d (i.e., vertices of the timeline of the depot). For depot $d \in \mathcal{D}$ we denote by $K^d(t) \subseteq K^d$ the set of duties covering trip $t \in \mathcal{T}$, furthermore, we denote by $K^d_-(i) \subseteq K^d$ ($K^d_+(i) \subseteq K^d$) the set of duties that contain a piece of work starting (ending) in vertex $i \in V^d$.

We also use two types of variables. First, we associate a *flow variable* y^d_{ij} with each depot-arc $(i,j) \in \bar{A}^d$ indicating the number of vehicles that cross arc (i,j). To ensure continuous attendance (Rule 5), and restricted changeover (Rule 6), the second type of our variables combines drivers and vehicles outside of a depot. Remember that a path between two vertices that correspond to relief points and using nondepot-arcs only represents a piece of work. From a different angle, such a path can be considered as a part of some vehicle block, that is why we can handle a piece of work as a driver-vehicle pair. That is, binary *duty variable* x^d_k indicates whether duty $k \in K^d$ is selected for depot $d \in \mathcal{D}$, if so, it means that a driver is assigned to duty k and for each piece of work of the duty a vehicle is assigned.

We also assign vehicle costs c^d_{ij} to each arc $(i,j) \in A^d$, and a working cost g^d_{ij} to each arc $(i,j) \in \tilde{A}^d$. With this, the driver cost (vehicle cost) of a piece of work is the cost of the corresponding path according to arc costs g^d_{ij} (c^d_{ij}), and the combined duty cost $\tilde{f}^d_k$ of duty $k \in K^d$ is the sum of the fixed driver cost, the vehicle cost of its pieces of work, and the working cost of its pieces of work. Now, we formulate the MDVCSP as:

$$\min \sum_{d\in\mathcal{D}} \sum_{(i,j)\in\bar{A}^d} c^d_{ij} y^d_{ij} + \sum_{d\in\mathcal{D}} \sum_{k\in K^d} \tilde{f}^d_k x^d_k \tag{7}$$

$$\sum_{d\in\mathcal{D}} \sum_{k\in K^d(t)} x^d_k = 1, \ \forall\, t \in \mathcal{T} \tag{8}$$

$$\sum_{k\in K^d_+(i)} x^d_k - \sum_{k\in K^d_-(i)} x^d_k = 0, \ \forall\, d \in \mathcal{D}, \forall\, i \in V^d \setminus \bar{V}^d \tag{9}$$

$$\sum_{(i,j)\in\bar{A}^d} y^d_{ij} + \sum_{k\in K^d_+(i)} x^d_k - \sum_{(j,i)\in\bar{A}^d} y^d_{ji} - \sum_{k\in K^d_-(i)} x^d_k = 0, \ \forall\, d \in \mathcal{D}, \forall\, i \in \bar{V}^d \tag{10}$$

$$0 \le y^d_{ij}, \ y^d_{ij} \in \mathbb{Z}, \ \forall\, d \in \mathcal{D}, \forall\, (i,j) \in \bar{A}^d \tag{11}$$

$$x^k_d \in \{0,1\}, \ \forall\, d \in \mathcal{D}, \forall\, k \in K^d. \tag{12}$$

The objective (7) minimizes the sum of vehicle and crew costs, as the fixed asset costs for the vehicles are built in the first term of (7), and all the other costs are contained in the second term of (7). Constraint set (8) ensures that each trip is covered by exactly one duty. Constraint sets (9)–(10) connect flow variables with the vehicle part of duty variables. That is, (9) specifies for a nondepot-vertex i that the number of pieces of work ending in vertex i (i.e., the number of vehicles arriving at vertex i) must be equal to the number of pieces of work starting in vertex i (i.e., the number of vehicles departing from vertex i). Constraint set (10) is analogous for depot-

vertices, but it takes into consideration that vehicles can wait in the depots. Note that flow variables are implicit integer, that is, they are always integer if duty variables are integer.

It is worth mentioning that in our formulation a duty variable (i.e., the corresponding column) contains only relevant information about the duty, namely, the start/end vertices of the piece(s) of work of the duty and the trips covered by the duty, if any. Notice that deadhead routes (e.g., routes between two consecutive trips) are not considered by the constraints. Moreover, the rules concerning the feasibility of duties do not appear explicitly in this formulation, only in the set K^d.

Note that limits on the number of vehicles in depots can be imposed by adding the constraints $y^d_{ts} \leq u^d$ to the model, where (t, s) is the circulation arc of the corresponding depot.

By construction, we have the following result.

Proposition 1 *Each optimal solution of the formulation (7)–(12) corresponds to an optimal solution for the MDVCSP, and each optimal solution for the MDVCSP is represented as an optimal solution for the formulation (7)–(12).*

5 Solution approach

In this section we present our solution method for the MDVCSP which is a branch-and-price procedure to solve *master problem* (7)–(12). That is, we compute a MIP containing just a few columns of the master problem (called *restricted master problem*) and perform a branch-and-bound procedure such that in each node of the search-tree we may add new columns (i.e., duties) to the LP-relaxation of the current restricted master problem.

More precisely, we create an initial restricted master problem (described in Section 5.1). We solve each node LP to optimality, that is, for each node we generate new duties until no one with a negative reduced cost is left as we describe in Section 5.2. At the root node we apply a two-stage approach. In the first stage we generate duties that contain one or two pieces of work starting and ending in the depot, and at the end of this stage we perform a primal solution search (described in Section 5.4). The reason for this is that with such a column set the constraints (9)–(10) are easy to satisfy, hence we expect that the search procedure can quickly find a good primal solution. In the second stage we generate duties without any limitations for their start and end locations, and we may also perform a primal solution search at the end of the stage. We describe our branching rules in Section 5.3. Our primary branching strategy is to assign trips to depots, and we use the SPP-based branching strategy as a secondary rule (if the primary rule failed to branch), and as a last resort, one may rely on the default branching strategy of the MIP solver.

5.1 Initial restricted master problem

The initial restricted master problem contains all of the flow variables and a set of initial duty variables that we create by obtaining a feasible solution for the MDVCSP by using a sequential

procedure. That is, we first formulate the MDVSP problem as a minimum cost multicommodity flow problem on the time-space networks using the given vehicle costs as in (Kliewer et al., 2006), and solve the MIP model with a standard software. Then, independently for each depot we create a set-partitioning formulation for the CSP (e.g., Freling et al. (2003)) to assign drivers to the obtained vehicle schedules. We solve the LP-relaxations of these problems with a column generation approach similar to the one we discuss in Section 5.2, then we solve the resulting restricted master problems with branch-and-bound, and use the solutions as initial column set for the MDVCSP.

Note, that if we failed to obtain feasible integer solution for any of the CSP problems, we could use fictive columns for the initial restricted master problem penalized by a high cost, or we could start branch-and-price with an initial restricted master problem containing no duty variables (see Farkas pricing in Section 5.2.3).

5.2 Pricing variables

Once the corresponding restricted master problem is solved we attempt to price out new variables (i.e., new duties) by using the dual information of the solution. Let λ_t ($t \in \mathcal{T}$) and μ_i^d ($i \in V^d$, $d \in \mathcal{D}$) be the dual variables associated to constraints (8) and (9)–(10), respectively.

To generate feasible duties we use a two-phase procedure similar to the one proposed by Freling et al. (1995a), that is, in the first phase we generate a set of feasible pieces by using a so-called piece generation network, and in the second phase we derive feasible duties. Since we generate pieces of work and duties independently for each depot, in the rest of this section we fix a depot $d \in \mathcal{D}$.

5.2.1 Generation of pieces of work

For each depot we derive a piece generation network from the corresponding time-space network consisting of all original arcs but depot-arcs, that is, the piece generation network for depot d is $\hat{D}^d = (V^d, A^d \setminus \bar{A}^d)$. We recall that each path in $\hat{D}^d$ between two vertices that correspond to relief points represents a piece of work. For a piece of work p let $A[p]$ and $\mathcal{T}[p]$ be the set of arcs and the set of trips covered by p, respectively, and let $s[p]$ and $e[p]$ be the start and the end vertex of p, respectively. The combined cost h of a piece of work p is the sum of vehicle and driver costs for all arcs covered by the piece of work, formally

$$h(p) := \sum_{ij \in A[p]} c_{ij}^d + g_{ij}^d.$$

The reduced cost $\hat{h}$ of a piece of work p (and the reduced cost of the corresponding path) is

$$\hat{h}(p) := h(p) - \mu_{s[p]}^d + \mu_{e[p]}^d - \sum_{t \in \mathcal{T}[p]} \lambda_t^d.$$

For the sake of efficiency, we do not generate all of the pieces of work, but obtain a set of feasible pieces by considering only the minimum reduced cost path between any two vertices in $\hat{D}^d$. To do this, we predetermine a processing order of vertices of $\hat{D}^d$ (which is a topological order in case the network is acyclic). By that, for any given node we can determine the shortest path arborescence in $O(|A|)$ time, thus we can determine the minimum reduced cost path for each pair of vertices in $O(|A||V|)$ total time.

At the root node of the search-tree it is clear that considering only the minimum reduced cost paths is sufficient in the sense that we will find at least one piece of work with negative reduced cost, if any. However, when branching decisions are to be considered this strategy may fail to find appropriate pieces. For example, assume that piece of work p corresponding to the minimum reduced cost path between vertices u and v is infeasible according to some of the branching decisions, but there is an another u–v path with negative reduced cost that admits a piece of work p' which is feasible according to all of the branching decisions. It is clear that we will fail to find the feasible piece of work p', since it is overshadowed by the infeasible piece of work p. That is why we should take branching decisions into consideration during piece or/and duty generation. We postpone the details until Section 5.3.

5.2.2 Generation of duties

Duties consisting of one piece of work can be easily generated by iterating over the previously obtained piece of work set. To generate combined duties (i.e., duties consisting of two pieces of work) we apply a straightforward pairing procedure using proper data structures and several acceleration techniques in order to avoid enumerating inherently infeasible pairs.

Once a set of pieces of work is obtained we build two *piece handler* data structures: $\mathcal{A}$ and $\mathcal{B}$. In piece handler $\mathcal{A}$ ($\mathcal{B}$) we create a *time block* for each time t in the time horizon containing pieces of work with sign-on start time (start time) t, and pieces of work in a block are sorted in increasing order according to their end time (sign-off end time).

In the following we describe how we generate feasible duties for a given combined duty type using the piece handlers. Briefly stated, we choose a piece of work as the first piece of the potential duty, then we enumerate the appropriate second pieces. The sketch of the procedure is presented in Algorithm 1.

For a combined duty type D let $D^{\min}_{start}$ and $D^{\max}_{start}$ be the earliest and latest sign-on start time, $D^{\max}_{end}$ be the latest sign-off end time, $D^{\min}_{piece}$ be the minimum piece length, $D^{\min}_{break}$ and $D^{\max}_{break}$ be the minimum and maximum break length, $D^{\max}_{working}$ be the maximum working time, $D^{\max}_{spread}$ be the maximum spread time. For a given piece of work p denote with p^{et} its end time; with p^{wt} its working time; and with p^{son} and p^{soff} its sign-on start time and sign-off end time, respectively.

First, we calculate an earliest and latest sign-on start time for the first piece of work (ES_f and LS_f, respectively) using the properties of duty type D. For example, we can simply use the

Algorithm 1 Generating combined duties

```
1: procedure GENERATEDUTIES(A, B, D)
2:     ES_f ← earliest sign-on start time for the first piece
3:     LS_f ← latest sign-on start time for the first piece
4:     for b_f in ES_f ... LS_f do
5:         for p_f in A[b_f] do
6:             ES_s ← earliest start time for the second piece
7:             LS_s ← latest start time for the second piece
8:             for b_s in ES_s ... LS_s do
9:                 LE_s ← latest sign-off end time for the second piece
10:                for p_s in B[b_s] do
11:                    if (p_f, p_s) is feasible according to type D and
               (p_f, p_s) is feasible according to branching decisions and
               (p_f, p_s) has negative reduced cost then
12:                        save duty (p_f, p_s)
13:                    end if
14:                end for
15:            end for
16:        end for
17:    end for
18: end procedure
```

trivial values $ES_f = D^{\min}_{start}$ and $LS_f = D^{\max}_{start}$, however we can easily strengthen LS_f such that:

$$LS_f = \min\left\{D^{\max}_{start}, D^{\max}_{end} - D^{\min}_{spread}\right\}.$$

After these values are calculated we iterate over the appropriate blocks of piece handler $\mathcal{A}$ (denoted by $\mathcal{A}[\cdot]$), and over the pieces of these blocks (line 4–5). For the actual first piece p_f we calculate an earliest and latest start time for the potential second piece of work (ES_s and LS_s, respectively). Again, we can use trivial bounds where we only take break length into consideration: $ES_s = p^{et}_f + D^{\min}_{break}$ and $LS_s = p^{et}_f + D^{\max}_{break}$, however we can use stricter bound for LS_s, that is

$$LS_s = \min\left\{p^{et}_f + D^{\max}_{break}, D^{\max}_{end} - D^{\min}_{piece}\right\}.$$

Then, we iterate over the appropriate blocks of piece handler $\mathcal{B}$, and over the pieces of these blocks (line 8 and 10). However, we can avoid to iterate over all the seconds pieces that constitute infeasible duty with the actual first piece. That is, we calculate a latest sign-off end time LE_s for the potential second piece, and if $p^{soff}_s > LE_s$ holds for the current second piece p_s then we can stop to iterate over the pieces of the current block, since pieces are sorted in increasing order according to their sign-off end time. For example, we can use the following value:

$$LE_s = \min\left\{D^{\max}_{end}, b_s + (D^{\max}_{working} - p^{wt}_f), D^{\max}_{spread} - p^{son}_f\right\},$$

where b_s is the common start time of pieces of work in the current block. Once we have a piece of work pair (p_f, p_s) we check whether it constitutes a feasible duty according to the given type, if so, we can check whether it is feasible according to branching decisions.

5.2.3 Farkas pricing

After branching is performed the restricted master problem of a new node may be infeasible due to fixings, but it does not mean that the master problem of the node is infeasible, so the node can not be pruned.

Again, one could resolve this issue by adding fictive columns to the LP penalized by a high cost, but instead, in such cases we perform a so-called Farkas pricing. That is, if the restricted master problem is infeasible we can obtain dual Farkas multipliers $\bar{\lambda}_t$ and $\bar{\mu}_v^d$ associated with constraints (8) and (9)–(10), respectively, to prove infeasibility according to the Farkas-Lemma. To make restricted master problem feasible we have to find a new column that violates this proof. It can be shown that this pricing problem is similar to the pricing problem for reduced cost pricing, but now we use a zero objective function and the dual Farkas multipliers instead of the original objective function and the dual solution. Thus, we can use the pricing method discussed in Section 5.2 with a minor modification to make the restricted master problem feasible.

5.3 Branching strategies

Now, we present our strategies to perform branch in a node where the optimal solution for the final restricted master problem is fractional. Remark, that flow variables y_{ij}^d are implicit integer, hence it is sufficient to consider only duty variables x_k^d in a branching rule.

5.3.1 Assign trips to depots

Our first branching strategies can be used in the multiple-depot case when there exists a trip that belongs to several depots in the current LP-relaxation. Formally, consider a fractional solution $(\bar{x}, \bar{y})$ to the relaxation of the corresponding restricted master problem, and let $C_{\bar{x}}(t, d)$ denote the commitment of trip t to depot d, that is

$$C_{\bar{x}}(t, d) := \sum_{k \in K^d(t)} \bar{x}_k^d.$$

If $0 < C_{\bar{x}}(t, d) < 1$ holds for a trip t and a depot d it means that trip t is committed to multiple depots according to solution $\bar{x}$. In this case we choose a trip $\bar{t}$ and a depot $\bar{d}$ such that $(\bar{t}, \bar{d}) = \arg\min_{(t,d)} |C_{\bar{x}}(t, d) - 0.5|$. We have two possibilities to perform branch on pair $(\bar{t}, \bar{d})$:

1. *Partitioning:* We create exactly two branches. We require to cover trip $\bar{t}$ by a duty from depot $\bar{d}$ on the one branch, and to cover by a duty from a depot that differs from $\bar{d}$ on the other branch. Formally,

$$\sum_{k \in K^{\bar{d}}(\bar{t})} x_k^{\bar{d}} = 1, \text{ binding branch} \tag{13}$$

$$\sum_{k \in K^{\bar{d}}(\bar{t})} x_k^{\bar{d}} = 0, \text{ banning branch.} \tag{14}$$

2. *Splitting:* Assume that trip $\bar{t}$ can be performed from depots $d_{i_1}, d_{i_2}, \ldots, d_{i_q}$. We create q branches, and force to cover trip $\bar{t}$ by a duty from depot d_{i_j} on the jth branch ($1 \leq j \leq q$). Formally, for the jth branch we have

$$\sum_{k \in K^{d_{i_j}}(\bar{t})} x_k^{d_{i_j}} = 1. \tag{15}$$

Note that these two branching rules are the same if we have exactly two depots.

As we remarked above, these branching rules are not complete in the sense that they cannot be used if each trip t is committed for a single depot, i.e., $C_{\bar{x}}(t, d) = 1$ holds for some depot d. However, handling these branching rules is quite easy without adding any inequalities of (13)–(14) or (15) to the problem. That is, on the one hand we can easily fix the appropriate existing variables to zero according to the corresponding branch. On the other hand, if a trip is forbidden to cover by a duty from the depot for which we want to price out new duties, we just erase the corresponding trip-arc from the piece generation network of the depot, and the pricing procedure described in Section 5.2 can be used without any modification.

5.3.2 SPP-based branching

This branching strategy is based on the branching scheme proposed by Ryan and Foster (1981) for problems with set partitioning structure, i.e., for MIPs of the form $\min\{wx \mid Ax = \mathbb{1}, x \in \{0,1\}^n\}$, where A is a $m \times n$ matrix with 0/1 columns representing some subsets of a set. Their branching scheme uses the observation that in every fractional solution of the LP-relaxation, there exists a pair of rows $(a_{i\cdot}, a_{j\cdot})$ with $0 < \sum_{c \in C(i,j)} x_c < 1$, where $C(i,j)$ is the set of columns covering both constraints $a_{i\cdot}$ and $a_{j\cdot}$, i.e., $C(i,j) = \{c \in \{1, \ldots, n\} \mid a_{ic} = a_{jc} = 1\}$. Their branching rule creates two branches: one forcing to cover rows $a_{i\cdot}$ and $a_{j\cdot}$ by the same column, and another one forcing to cover the two rows by different columns.

In our branching strategy, we branch on duty variables utilizing the set partitioning structure of constraints (8). Consider a fractional solution $(\bar{x}, \bar{y})$ to the relaxation of the corresponding restricted master problem, and for trips t, u and depot d let

$$B_{\bar{x}}(t, u; d) := \sum_{k \in K^d(t,u)} \bar{x}_k^d,$$

where $K^d(t,u) \subset K^d$ is the set of duties covering both trips t and u. We select a pair of trips $(\bar{t},\bar{u})$ and a depot $\bar{d}$ to branch on such that $(\bar{t},\bar{u};\bar{d}) = \arg\min_{(t,u;d)} |B_{\bar{x}}(t,u;d) - 0.5|$. The branching scheme requires to cover trips $\bar{t}$ and $\bar{u}$ by the same duty from depot d on one branch and not to cover by the same duty from depot d on the other. Formally,

$$\sum_{k \in K^{\bar{d}}(\bar{t},\bar{u})} x_k^d = 1, \text{ same branch} \tag{16}$$

$$\sum_{k \in K^{\bar{d}}(\bar{t},\bar{u})} x_k^d = 0, \text{ diff branch.} \tag{17}$$

Note that this branching rule can be used for both of the single-depot and the multiple-depot case if there exists trips t, u and a depot d such that $0 < B_{\bar{x}}(t,u;d) < 1$.

Again, we do not intend to add any of the inequalities (16)–(17) to the restricted master problem, however, handling this branching rule in the pricing procedure is a bit cumbersome as we explain in the following. Assume that in a node we would like to generate new feasible duties for a given depot, but a branching decision requires not to cover trips t and u by the same duty. In addition, assume that a combined duty consisting of pieces of work p_t and p_u has a negative reduced cost, where pieces of work p_t and p_u contain trips t and u, respectively. This duty is infeasible according to the branching decision, and it may shadow a feasible duty with negative reduced cost. Thus we have to ensure that pieces of work (i) containing trip t, (ii) not containing trip t, (iii) containing trip u, (iv) not containing trip u are also generated. These terms are going to be more complicated in nodes with higher depth. In order to resolve this difficulties we apply a two-step procedure. That is, in the first step we generate duties as we described before until no more duties with negative reduced cost are left. If in the last pricing round we do not refuse any duties according to branching decisions, we can stop (i.e., the node is solved to optimality), since no overshadowed duties with negative reduced costs are left. Otherwise, in the second step we choose a duty which was refused in the last pricing round and generate all duties that may be overshadowed by this duty. More specifically, assume that the refused duty consists of pieces of work p_1 and p_2 where p_i refers to an u_i–v_i path for $i = 1, 2$, respectively. We construct a piece of work set S by generating all pieces of work that correspond to an u_i–v_i path ($i = 1, 2$). To generate duties in the second step we use the piece of work set corresponding to the shortest paths along with the piece of work set S. We repeat this procedure until no duties with negative reduced cost are left or refused.

5.3.3 Default 0-1 branching

As we mentioned before, when all of our strategies failed to branch, as a last resort we rely on the default branching strategy of the MIP solver. That is, a fractional duty variable x_k^d is chosen, and it is forced to 0 on the left branch and 1 on the right branch. In the former case we need to ensure that this forbidden duty will be not regenerated during the pricing procedure. Thus, for

each node we maintain a list of forbidden duties and once a potential duty occurs in the pricing procedure (see Algorithm 1, line 11) we check whether it is in the list, and if so, we reject that duty since it is infeasible according to branching decisions.

5.4 Primal solution search

Any time during the solution method we can perform an obvious primal solution search approach, that is we solve problem (7)–(12) with the current column set. However, such a problem can be hard to solve, so it is not worth to apply this method frequently.

6 Computational results

In this section we present our computational results.

6.1 Test environment and implementation

All the computational experiments were performed on a workstation with 4GB RAM, and XEON X5650 CPU of 2.67 GHz, and under Linux operating system. All experiments were run using a single thread only.

Our solution method was implemented in C++ programming language using SCIP Optimization Suite (version 3.1.1) (Achterberg, 2009) as a branch-and-price framework. We also used FICO Xpress Optimization Suite (version 28.01.09) (FICO, 2008) callable library (Xpress) to solve certain phases. To handle graphs and to perform network algorithms we used the LEMON C++ library (version 1.3.1) (Dezső et al., 2011).

6.2 Instances and problem parameters

We tried to comply with Steinzen et al. (2010) as much as possible, that is, we used the same instance set, the same duty parameters and the same costs as in (Steinzen et al., 2010).

We used the randomly generated problem instances of Huisman available in (Huisman, 2003) and described in (Huisman et al., 2005). These instances are classified into two classes according to travel speed (i.e., length of the trips), that is, class A consists of shorter trips than class B, hence vehicle blocks and duties cover more trips, thus instances in class A can be considered more difficult. In class A for each $n = 80, 100, 160, 200, 320$ there are 10 instances (one trip-file and one deadhead-file) containing n trips and requiring 4 depots and 4 or 5 stations.

In accordance with Huisman (2004) we used five types of duties with the properties described in Table 2. A *tripper duty* consists of one piece of work with length between 30 minutes and

Table 2 Properties of duty types

	Tripper		Early		Day		Late		Split	
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
start time					8:00		13:15			
end time				16:30		18:14				19:30
piece length	0:30	5:00	0:30	5:00	0:30	5:00	0:30	5:00	0:30	5:00
break length	-	-	0:45		0:45		0:45		1:30	
spread time				9:45		9:45		9:45		12:00
working time				9:00		9:00		9:00		9:00

5 hours, while the combined duties (*early*, *day*, *late*, *split*) contains exactly two pieces of work separated by a break. For duties starting (ending) in a depot we assessed a sign-on (sign-off) time of 10 (of 5) minutes, and for duties starting (ending) at a station we assessed a sign-on (sign-off) time of 15 minutes plus the deadhead time between the start (end) station and the depot. Start and end times in Table 2 correspond to the sign-on start and sign-off end time of the duty, respectively.

We assigned a fixed cost of 1000 for each vehicle and a cost of 1 for each minute a vehicle is outside of the depot. We assigned a fixed cost of 1000 for each duty and a cost of 0.1 for each minute a driver is working.

6.3 Running details

We solved problems with gap limit set to 0.5%, and time limit set to $20 \times |\mathcal{T}|$ seconds, i.e., the solution process could be stopped due to three reasons: (i) the best solution was proven to be optimal, (ii) the gap limit was reached (i.e., the relative gap between the lower bound and the current best solution was at most 0.5%), (iii) the time limit was reached (i.e., the execution time exceeded $20 \times |\mathcal{T}|$ seconds).

As we mentioned in Section 5, at the root node we used a two-stage approach for generating duties. At the end of the first stage we applied our primal heuristic, that is, we called Xpress with time limit set to 60 seconds to solve the current restricted master problem. At the end of the second stage we applied this heuristic only if the number of variables did not exceed 30 000.

6.4 Experiments

6.4.1 Branching rules

In these experiments we aimed to compare the two branching strategies described in Section 5.3.1. In order to make a more extensive experiment we matched all of the trip-files with all of the

Table 3 Summary of experiments on branching rules

Problem	Rule	Status			Bound			Best solution			Time
		O	G	T	Lower	Upper	Gap	v	d	v+d	
80A	Partitioning	9	50	41	34 772.2	35 410.8	1.8%	9.5	18.5	28.0	755.6
	Splitting	8	49	43	34 769.4	35 481.4	2.0%	9.5	18.6	28.0	777.5
100A	Partitioning	6	49	45	41 624.0	42 464.2	2.0%	11.4	22.1	33.5	1136.6
	Splitting	5	43	52	41 621.7	42 533.2	2.2%	11.4	22.1	33.6	1205.4

deadhead-files for these tests, i.e., we used $10 \times 10 = 100$ problem instances. In Table 3 we present our results where we indicate the summarized solution status (these are, the number of instances that solved to optimality (O), the number of instances where gap limit was reached (G), the number of instances where solving process was stopped due to time limit (T)); the best lower (Lower) and upper bound (Upper) and the corresponding gap (Gap) which is calculated as $100 \times (\text{Upper} - \text{Lower})/\text{Lower}$; the number of vehicles (v) and the number of drivers (d) in the best solution; and the execution time in seconds (Time).

Both for 80-trip and 100-trip instances, the partitioning based branching strategy gave the best results in terms of execution time, and the quality of solutions as well. Moreover, more instances were solved within the time limit with that rule. According to these results, in the following experiments we used the partitioning rule as the primary branching strategy.

6.4.2 Evaluation of the integrated method

In Tables 4 and 5 we present the evaluation of our integrated method on 80-trip and 100-trip instances, respectively. In these table we indicate the solution status (optimal: the instance is solved to optimality; gap/time limit: the solving process is stopped due to the gap/time limit was reached); the lower bound at the root node (Root), and at the end of the procedure (Global); the value of the best solution (Upper bound); the corresponding gap (Gap) which is calculated as $100 \times (\text{Upper bound} - \text{Global})/\text{Global}$; and the execution time in seconds (Time).

We can see that 2 out of ten 80-trip instances are solved to optimality, and 3 other instances are solved with gap less than 0.5%, moreover, the average gap of the 80-trips instances is 2.5%. For the 100-trip instances we also solved 2 out of ten instances to optimality, and 4 more instances are solved with gap limit, while the average gap is 2.2%.

We also remark that most of the computation time was spent at the root node for finding the optimal LP solution which sometimes required the generation of thousands of columns. In the other tree nodes, finding the optimum solution took much less effort in general.

Table 4 Summary of the evaluation of the integrated method on problem set 80A

Problem	Status	Lower bound		Upper bound	Gap	Time
		Root	Global			
1	gap limit	31 619.6	31 619.6	31 702.3	0.3%	88.6
2	time limit	27 497.8	27 498.3	29 079.7	5.8%	1602.5
3	optimal	32 750.7	32 750.7	32 750.7	0.0%	87.1
4	time limit	34 162.4	34 169.8	34 922.2	2.2%	1600.6
5	gap limit	32 175.4	32 175.4	32 188.6	0.0%	112.0
6	time limit	31 393.9	31 407.5	32 879.4	4.7%	1602.6
7	gap limit	36 133.7	36 133.7	36 266.6	0.4%	115.9
8	time limit	43 017.6	43 040.9	44 419.3	3.2%	1601.1
9	optimal	34 638.4	34 638.4	34 643.9	0.0%	734.0
10	time limit	42 583.6	42 619.4	45 716.3	7.3%	1601.3
average		34 597.3	34 605.4	35 456.9	2.5%	914.6

Table 5 Summary of the evaluation of the integrated method on problem set 100A

Problem	Status	Lower bound		Upper bound	Gap	Time
		Root	Global			
1	optimal	49 183.8	49 183.8	49 183.8	0.0%	390.8
2	time limit	41 311.8	41 326.8	43 552.4	5.4%	2002.3
3	time limit	35 896.6	35 910.3	38 519.5	7.3%	2000.8
4	gap limit	40 217.2	40 217.2	40 255.5	0.1%	175.1
5	optimal	45 424.8	45 424.8	45 424.8	0.0%	344.5
6	gap limit	35 543.3	35 543.3	35 543.8	0.0%	230.0
7	time limit	36 242.3	36 257.3	37 231.3	2.7%	2003.2
8	gap limit	45 403.5	45 403.5	45 453.4	0.1%	237.1
9	time limit	50 566.0	50 572.6	53 708.4	6.2%	2002.7
10	gap limit	33 912.2	33 912.2	34 001.5	0.3%	683.7
average		41 370.2	41 375.2	42 287.4	2.2%	1007.0

6.4.3 Comparison of methods

In Table 6 we summarize our comparison of sequential and integrated methods, while the detailed results of our experiments can be found in Tables 7 and 8. Method Seq. refers to the sequential approach we used to obtain in the initial restricted master problem, while the next two methods refer to our integrated approach. In case of method Int. (first) we interrupted the solution procedure right after we found a feasible solution to the problem. In case of method Int. (timelimit) we interrupted our procedure only when the time limit was reached (or we found a good enough solution). Method Int. (Steinzen et al., 2010) refers to the integrated approach of Steinzen et al. (2010) which was tested on a Dell OptiPlex GX620 personal computer with an Intel Pentium IV

Table 6 Comparing sequential and integrated methods

Problem	Method	v	d	v+d	Cost	Time
80A	Seq.[a]	9.2	24.3	33.5	40 588.0	1.2
	Int. (first)[a]	9.6	18.6	28.2	35 668.5	4.1
	Int. (timelimit)[a]	9.5	18.5	28.0	35 456.9	914.6
	Int. (Steinzen et al., 2010)[b]	9.2	19.1	28.2		235.0
100A	Seq.	11.0	28.2	39.2	47 792.7	1.6
	Int. (first)	11.4	22.0	33.4	42 428.5	31.8
	Int. (timelimit)	11.4	21.9	33.3	42 287.4	1007.0
	Int. (Steinzen et al., 2010)	11.0	22.7	33.7		369.0

[a] tested on a workstation with 4GB RAM, and XEON X5650 CPU of 2.67 GHz, and under Linux operating system.

[b] tested on a Dell OptiPlex GX620 personal computer with an Intel Pentium IV 3.4 GHz processor and 2 GB of main memory under Windows XP.

3.4 GHz processor under Windows XP. In this table we indicate the number of vehicles (v), the number of drivers (d), the cost of the best solution (Cost); and the execution time in seconds (Time). Note that in Table 6 in case of method Int. (Steinzen et al., 2010) we do not indicate the solution costs, and in Tables 7 and 8 we do not indicate the results of method Int. (Steinzen et al., 2010) since these are not provided in (Steinzen et al., 2010). We contacted the authors, however, they could not provide these detailed results.

On the one hand, our experiments re-proved that one can obtain better solutions using the integrated approach instead of the sequential method. On the other hand, observe that we could improve on the first integer solution if we run the procedure until a time limit or a gap limit is reached, however, the average improvement over the first integer feasible solution is 1.1% in the 80-trip case, and 0.3% in the 100-trip case.

One can see that our integrated method found solutions with fewer vehicles plus drivers than Steinzen et al. (2010). Both for 80-trip and 100-trip instances, our method found the first integer solution quickly in 4.1 and 31.8 seconds, respectively, and on average it was at least as good as the final solution of Steinzen et al. (2010). We also note that Steinzen et al. (2010) presented computational results for instances with $n = 160, 200, 320, 400, 640$, as well, however, solving instances with 160 trips took already about 1600 seconds on average, while 640-trip instances required about 16 hours. We also made experiments on the 160-trip instances, however, we were not able to solve any of these instances neither to optimality, nor with gap limit, in fact, the column generation procedure at the root node required more than 3 hours on average. Our best solutions yielded 11.5% gap on average, and the average number of vehicles and drivers used in these solutions ($v + d = 50.5$) is worse than that of Steinzen et al. (2010) ($v + d = 46.6$).

Table 7 Comparing sequential and integrated methods on problem set 80A

Problem	Method	v	d	v+d	Cost	Time
1	Seq.	9	21	30	36 525.2	0.4
	Int. (first)	9	16	25	31 702.3	1.4
	Int. (timelimit)	9	16	25	31 702.3	88.6
2	Seq.	7	19	26	31 747.5	2.4
	Int. (first)	8	15	23	29 079.7	10.7
	Int. (timelimit)	8	15	23	29 079.7	1602.5
3	Seq.	9	28	37	43 155.6	0.4
	Int. (first)	9	17	26	32 750.7	1.2
	Int. (timelimit)	9	17	26	32 750.7	87.1
4	Seq.	9	22	31	38 551.5	0.6
	Int. (first)	9	18	27	34 922.2	2.7
	Int. (timelimit)	9	18	27	34 922.2	1600.6
5	Seq.	9	29	38	43 364.7	1.4
	Int. (first)	9	17	26	32 188.6	1.6
	Int. (timelimit)	9	17	26	32 188.6	112.0
6	Seq.	8	21	29	35 747.4	2.3
	Int. (first)	9	17	26	32 879.4	4.9
	Int. (timelimit)	9	17	26	32 879.4	1602.6
7	Seq.	10	25	35	41 946.5	0.9
	Int. (first)	10	19	29	36 283.1	0.9
	Int. (timelimit)	10	19	29	36 266.6	115.9
8	Seq.	11	28	39	48 293.9	0.6
	Int. (first)	11	24	35	44 419.3	1.4
	Int. (timelimit)	11	24	35	44 419.3	1601.1
9	Seq.	9	22	31	38 355.7	1.8
	Int. (first)	10	19	29	36 742.9	13.6
	Int. (timelimit)	9	18	27	34 643.9	734.0
10	Seq.	11	28	39	48 191.6	1.0
	Int. (first)	12	24	36	45 716.3	2.7
	Int. (timelimit)	12	24	36	45 716.3	1601.3
average	Seq.	9.2	24.3	33.5	40 588.0	1.2
	Int. (first)	9.6	18.6	28.2	35 668.5	4.1
	Int. (timelimit)	9.5	18.5	28.0	35 456.9	914.6

7 Conclusions

In this paper we have devised an exact method for solving the integrated multiple-depot vehicle and crew scheduling problem optimally. Our computational results show that with limited computational resources (computation time + single CPU thread), nearly optimal schedules can be

Table 8 Comparing sequential and integrated methods on problem set 100A

Problem	Method	v	d	v+d	Cost	Time
1	Seq.	13	30	43	53 098.0	1.2
	Int. (first)	13	27	40	50 243.2	62.1
	Int. (timelimit)	13	26	39	49 183.8	390.8
2	Seq.	11	29	40	48 977.1	0.9
	Int. (first)	12	22	34	43 552.4	61.5
	Int. (timelimit)	12	22	34	43 552.4	2002.3
3	Seq.	9	25	34	41 973.9	0.8
	Int. (first)	11	19	30	38 519.5	62.2
	Int. (timelimit)	11	19	30	38 519.5	2000.8
4	Seq.	11	28	39	47 042.1	1.1
	Int. (first)	11	21	32	40 255.5	4.3
	Int. (timelimit)	11	21	32	40 255.5	175.1
5	Seq.	12	29	41	50 366.5	1.1
	Int. (first)	12	24	36	45 630.5	4.1
	Int. (timelimit)	12	24	36	45 424.8	344.5
6	Seq.	10	23	33	40 405.2	1.3
	Int. (first)	10	18	28	35 579.0	2.4
	Int. (timelimit)	10	18	28	35 543.8	230.0
7	Seq.	10	23	33	40 954.1	3.2
	Int. (first)	10	19	29	37 231.3	65.5
	Int. (timelimit)	10	19	29	37 231.3	2003.2
8	Seq.	12	29	41	50 224.6	1.1
	Int. (first)	12	24	36	45 563.4	3.5
	Int. (timelimit)	12	24	36	45 453.4	237.1
9	Seq.	13	34	47	57 771.2	2.8
	Int. (first)	14	28	42	53 708.4	45.4
	Int. (timelimit)	14	28	42	53 708.4	2002.7
10	Seq.	9	32	41	47 113.8	2.6
	Int. (first)	9	18	27	34 001.5	7.1
	Int. (timelimit)	9	18	27	34 001.5	683.7
average	Seq.	11.0	28.2	39.2	47 792.7	1.6
	Int. (first)	11.4	22.0	33.4	42 428.5	31.8
	Int. (timelimit)	11.4	21.9	33.3	42 287.4	1007.0

found for problems with 80–100 trips and 4 depots. In order to increase the problem size, one possible direction is to exploit multiple CPU cores/threads, but for that, one needs a parallel branch-and-price solver. Currently, the parallel branch-and-price implementation of SCIP is at

the conceptual stage. Another option would be to get lower bounds faster, for which further acceleration strategies are needed.

Acknowledgements This work has been supported by the OTKA grant K112881, and by the GINOP-2.3.2-15-2016-00002 grant of the Ministry of National Economy of Hungary. The authors are grateful to the developers of the SCIP Optimization Suite for their support.

References

T. Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. `http://mpc.zib.de/index.php/MPC/article/view/4`.

M. Ball, L. Bodin, and R. Dial. A matching based heuristic for scheduling mass transit crews and vehicles. *Transportation Science*, 17(1):4–31, 1983.

J. Balogh and J. Békési. Driver scheduling for vehicle schedules using a set covering approach: a case study. In *Proceedings of the 10th International Conference on Applied Informatics, Accepted*, 2014.

J. Békési, A. Brodnik, M. Krész, and D. Pash. An integrated framework for bus logistics management: Case studies. *Logistik Management*, pages 389–411, 2009.

A. A. Bertossi, P. Carraresi, and G. Gallo. On some matching problems arising in vehicle scheduling models. *Networks*, 17(3):271–281, 1987.

L. Bodin, B. Golden, and A. Assad. Routing and scheduling of vehicles and crews: the state of the art. *Computers & Operations Research*, 10(2):63–211, 1983.

R. Borndörfer, A. Löbel, and S. Weider. A bundle method for integrated multi-depot vehicle and duty scheduling in public transit. *Computer-Aided Systems in Public Transport*, pages 3–24, 2008.

S. Bunte and N. Kliewer. An overview on vehicle scheduling models. *Public Transport*, 1(4): 299–317, 2009.

K. Darby-Dowman, J. Jachnik, R. Lewis, and G. Mitra. Integrated decision support systems for urban transport scheduling: Discussion of implementation and experience. In *Computer-Aided Transit Scheduling*, pages 226–239. Springer, 1988.

B. Dezső, A. Jüttner, and P. Kovács. LEMON–an open source C++ graph template library. *Electronic Notes in Theoretical Computer Science*, 264(5):23–45, 2011.

J. Falkner and D. Ryan. Express: Set partitioning for bus crew scheduling in christchurch. In *Computer-Aided Transit Scheduling*, pages 359–378. Springer, 1992.

FICO. Xpress optimization suite, 2008.

M. Fischetti, S. Martello, and P. Toth. The fixed job schedule problem with spread-time constraints. *Operations Research*, 35(6):849–858, 1987.

M. Fischetti, S. Martello, and P. Toth. The fixed job schedule problem with working-time constraints. *Operations Research*, 37(3):395–403, 1989.

R. Freling, C. G. E. Boender, and J. M. P. Paixão. An integrated approach to vehicle and crew scheduling. Technical Report 9503/A, Econometric Institute, Erasmus University Rotterdam, Rotterdam, The Netherlands, 1995a.

R. Freling, J. M. P. Paixão, and A. P. Wagelmans. *Models and algorithms for vehicle scheduling*. Econometric Institute, Erasmus University Rotterdam Rotterdam, The Netherlands, 1995b.

R. Freling, A. P. Wagelmans, and J. M. P. Paixão. Models and algorithms for single-depot vehicle scheduling. *Transportation Science*, 35(2):165–180, 2001.

R. Freling, D. Huisman, and A. P. Wagelmans. Models and algorithms for integration of vehicle and crew scheduling. *Journal of Scheduling*, 6(1):63–85, 2003.

C. Friberg and K. Haase. An exact branch and cut algorithm for the vehicle and crew scheduling problem. In *Computer-Aided Transit Scheduling*, pages 63–80. Springer, 1999.

A. Gaffi and M. Nonato. An integrated approach to ex-urban crew and vehicle scheduling. In *Computer-Aided Transit Scheduling*, pages 103–128. Springer, 1999.

V. Gintner, N. Kliewer, and L. Suhl. A crew scheduling approach for public transit enhanced with aspects from vehicle scheduling. In *Computer-Aided Systems in Public Transport*, pages 25–42. Springer, 2008.

K. Haase, G. Desaulniers, and J. Desrosiers. Simultaneous vehicle and crew scheduling in urban mass transit systems. *Transportation Science*, 35(3):286–303, 2001.

D. Huisman. Random data instances for multiple-depot vehicle and crew scheduling, 2003. `http://people.few.eur.nl/huisman/instances.htm`.

D. Huisman. *Integrated and Dynamic Vehicle and Crew Scheduling*. PhD thesis, Erasmus School of Economics (ESE), 2004.

D. Huisman, R. Freling, and A. P. Wagelmans. Multiple-depot integrated vehicle and crew scheduling. *Transportation Science*, 39(4):491–502, 2005.

N. Kliewer, T. Mellouli, and L. Suhl. A time–space network based exact optimization model for multi-depot bus scheduling. *European Journal of Operational Research*, 175(3):1616–1627, 2006.

B. Laurent and J.-K. Hao. Simultaneous vehicle and crew scheduling for extra urban transports. In *New Frontiers in Applied Artificial Intelligence*, pages 466–475. Springer, 2008.

M. Mesquita and A. Paias. Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem. *Computers & Operations Research*, 35(5):1562–1575, 2008.

M. Mesquita, A. Paias, and A. Respício. Branching approaches for integrated vehicle and crew scheduling. *Public Transport*, 1(1):21–37, 2009.

I. Patrikalakis and D. Xerocostas. A new decomposition scheme of the urban public transport scheduling problem. In *Computer-Aided Transit Scheduling*, pages 407–425. Springer, 1992.

A.-S. Pepin, G. Desaulniers, A. Hertz, and D. Huisman. Comparison of heuristic approaches for the multiple depot vehicle scheduling problem. Technical report, Econometric Institute Research Papers, 2006.

D. M. Ryan and B. A. Foster. An integer programming approach to scheduling. *Computer scheduling of public transport urban passenger vehicle and crew scheduling*, pages 269–280, 1981.

D. Scott. A large scale linear programming approach to the public transport scheduling and costing problem. In *Computer Scheduling of Public Transport 2*, pages 473–491. Elsevier, 1985.

I. Steinzen. *Topics in integrated vehicle and crew scheduling in public transit.* PhD thesis, University of Paderborn, 2007.

I. Steinzen, V. Gintner, L. Suhl, and N. Kliewer. A time-space network approach for the integrated vehicle-and crew-scheduling problem with multiple depots. *Transportation Science*, 44(3):367–382, 2010.

E. Tosini and C. Vercellis. An interactive system for extra-urban vehicle and crew scheduling problems. In *Computer-Aided Transit Scheduling*, pages 41–53. Springer, 1988.